UNIVERSIDAD VERACUZANA
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRONICA
BOCA DEL RIO, VERACRUZ

PROGRAMA EDUCATIVO
INGENIERÍA INFORMÁTICA

EXPERIENCIA EDUCATIVA
INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

DOCENTE
DR. LUIS FELIPE MARÍN URÍAS

ESTUDIANTE
SAÚL ROMERO PRADO

FECHA DE ENTREGA
14/12/2021

¡Huye Kemonito!

Introducción

Los agentes inteligentes son aquellos que son capas de percibir un entorno y tomar una decisión dependiendo de las circunstancias de su entorno. Una de las formas en las que se puede representar el pensamiento de una agente es calculando la ruta de un unto a otro. Esta forma es con la que trabaja Google, por ejemplo, así es como nos da las rutas dentro de Maps.

A lo largo del curso de Introducción a la inteligencia Artificial se analizaron diferentes algoritmos con los que un agente puede recorrer un entorno, desde los algoritmos con información como lo son el Greedy o el A-Estrella hasta aquellos que trabajan sin más información que su origen y destino.

El objetivo principal del proyecto que se detallara en este documento es demostrar las diferencias de eficiencia y forma de ejecutar 4 algoritmos diferentes para un mismo objetivo. Los cuatro algoritmos elegidos fueron Greedy, A-Estrella, Búsqueda Primero en Profundidad (BPP) y Búsqueda Primero en Anchura (BPA), se eligieron dos que usan información y dos que no lo hacen con la meta de que las diferencias ala hora de tomar decisiones sea notable y así se pueda apreciar el cómo sin importar que todos hacen lo mismo cada uno lo hace a su manera.

Implementación

La implementación de estos algoritmos está hecha en el entorno de desarrollo de Godot utilizando como lenguaje de programación el mismo que ya nos brinda en entorno el cual es GDScript.

REPRESENTACION DEL ENTORNO

Como primer paso para esta implementación se pensó en como representar el entorno para que la computadora pudiera entenderlo, hasta ese momento aun no llegaba la clase donde se analizaron métodos que se ocupan, sin embargo, se logró encontrar una forma un poco silvestre pero funcional de representar nuestro entorno en Godot.

La forma elegida para representar el entorno fue mediante un sistema de esquinas el cual consiste básicamente en mapear cada una de las esquinas dentro de la pantalla y establecer las conexiones que tenían entre sí, para así poder ir de una esquina a otra. A continuación, los pasos que se siguieron para la representación de del entorno:

1. Se dibuja el mapa 2D de tal forma que todas las esquinas se conecten al menos con alguna otra, esto para poder movernos a cualquier lugar disponible.
2. Ya con el mapa dibujado se procede a encontrar la posición que cada esquina ocupa en el espacio, esto es de vital importancia y de debe hacer con cautela, ya que una esquina mal posicionada puede causar problemas a la hora de identificar los espacios libres.
3. Una vez cada esquina este identificada, se identifica cada punto de salida que se tiene a partir de esa esquina.
4. Esa información se representa dentro de GDScript como un diccionario que emula un grafo, donde las llaves son cada posición de nuestras esquinas y los valores son un arreglo de las conexiones que tiene cada una de las esquinas.
5. Para eso se creó una función la cual cada que se manda a llamar nos regresa un grafo base con las esquinas y sus conexiones.
6. Y así es como se obtiene la representación de nuestro entorno.

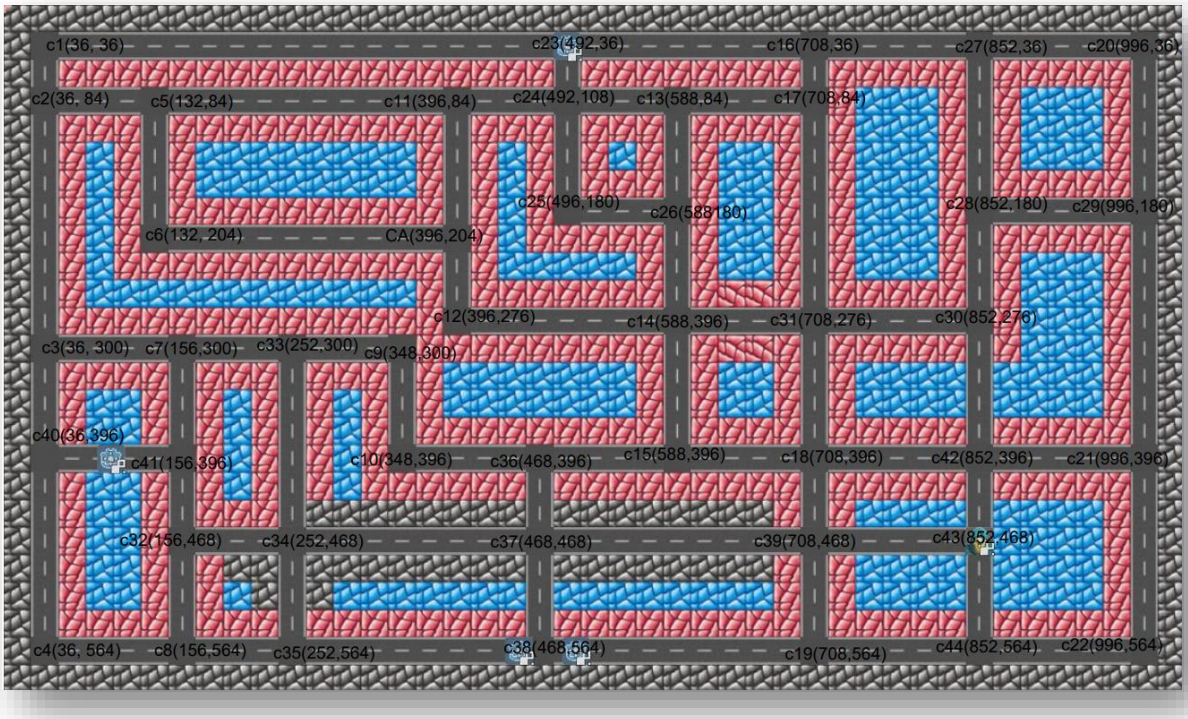

Ilustración 1: representación gráfica del mapeado del entorno.

¿DONDE ESTOY?

Antes de proceder a trabajar con los algoritmos es importante saber donde esta cada agente, ya que el simple hecho de las coordenadas no les sirve a los algoritmos para calcular la ruta.

Para esto se implementa una función la cual se encarga de leer el grafo y determinar si el agente se encuentra en una esquina y si no es así entre que par de esquinas se encuentra este.

Si el caso es que se encuentra en una esquina nuestro punto de partida será esa esquina por lo cual no será necesario más que eso.

Por el contrario, ni no estamos en una esquina y tenemos que determinar las esquinas entre las que se encuentra esto afectará a nuestro grafo, ya que la ubicación de nuestro agente se convertirá en una esquina mas del grafo que tendrá como esquenas vecinas a las esquinas entre las cuales se encontraba.

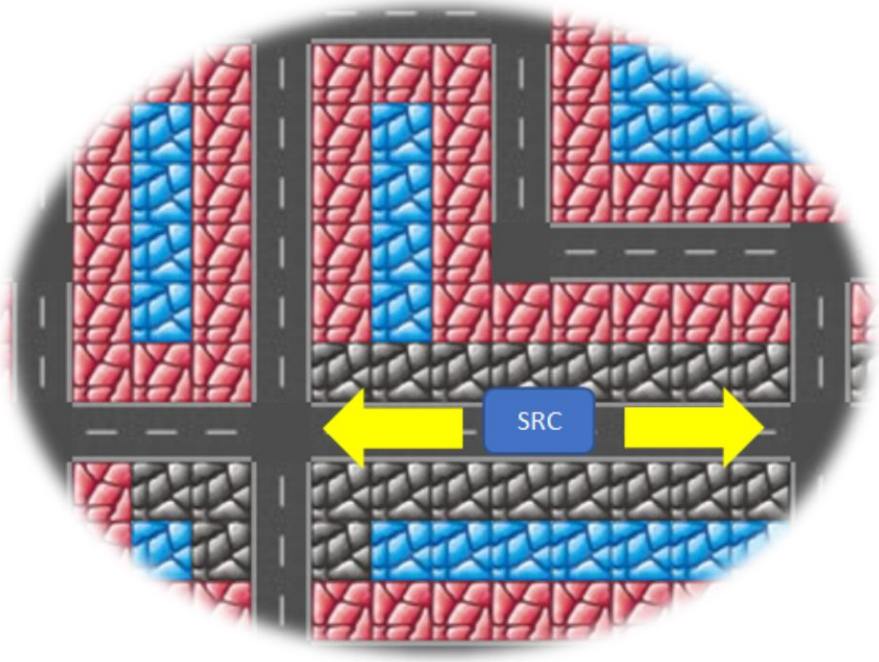

Ilustración 2: integración del agente al entorno.

ALGORITMOS DE BSUQUEDA DE CAMINO

Una vez que ya tenemos la representación de nuestro entorno, ahora procederemos a aprender a movernos sobre este, para eso se ocupan los cuatro algoritmos antes mencionados, Para implementar cada uno de estos, se ocupa una clase para cada uno, y con esta clase nosotros podremos construir nuestro punto de partida y nuestro punto de llegada para que esta nos regrese la meta conectada a cada punto que se necesito visitar para llegar a ella.

Greedy

Este algoritmo es uno de los dos que contienen información para encontrar un camino, este se ayuda de la distancia que tiene cada una de sus esquinas vecinas con respecto a la meta y así va iterando y decidiendo por donde moverse hasta encontrar la meta y regresar el camino. Entonces, aunque no es la mejor forma es bastante efectivo, a continuación, se muestra de manera abstracta el cómo trabaja la clase que lo hace.

- Para instanciar un Nodo Greedy se necesita una posición de origen, una posición de destino, un valor padre que para nuestro nodo raíz será NULL, el grafo base, y dos listas que nos informan los elementos visitados y por visitar, en la raíz estas están vacías siempre.
- Esta clase cuenta con método llamado “expande”, este trabaja de manera recursiva y nos regresa el camino.
- Expande nos realiza todo el proceso que nos dicta el algoritmo greedy, va preguntando nodo por nodo hasta encontrar la meta eso siempre basándose en la heurística establecida que es la distancia.
- Gracias al algoritmo implementado expande nodo no tendrá que mantener una pila de recursión muy grande, ya que gracias a nuestra heurística se disminuye el número.
- Al final, esta nos regresa la meta conectada con el camino seguido para encontrarla.

A-Estrella

Este algoritmo es básicamente un Greedy+ ya que utiliza la misma técnica de recorrimiento, pero esta le agrega un plus, ya que contempla el costo que tendrá ir a ciertos puntos, eso se suma a la heurística que ya teníamos y nos da una precisión mucho mayor y tendrá una posibilidad mayor de encontrar la mejor ruta. A continuación, los puntos importantes para el funcionamiento de la clase del Nodo A-Estrella:

- Esta clase es muy parecida a la del greedy, para crear una instancia de esta se ocupan los mismos parámetros que para el greedy pero este también necesita un costo, el cual para nuestro nodo raíz debe de ser 0 ya que el costo para llegar ahí es nulo.
- Al igual que el greedy está también contiene un método “expande” el cual cumple exactamente la misma función, pero implementada con le algoritmo A-Estrella.
- Expande se encarga de realizar toda la recursión y los cuestionamientos necesarios para encontrar el camino.
- El expande de A-Estrella es más complejo por ende procesa un poco más, pero todo es valido para poder obtener la mejor ruta posible.

- Al igual que greedy el tener información para tomar la decisión reduce en gran número las recursiones posibles.

Búsqueda Primero en Profundidad (BPP)

Este algoritmo es uno muy famoso a la hora de las búsquedas en árboles, es bastante simple, no ocupa información y es uno de los que se suele aprender primero, sin embrago esta forma de búsqueda es bastante engorrosa, aunque el algoritmo es simple, los resultados no siempre son los mejores. A pesar de no encontrar la mejor ruta casi siempre, lo importante es que sea como sea la encuentra. A continuación, se redacta abstractamente como se compone la clase que implementa este algoritmo:

- Para instanciar la clase que implementa el algoritmo BPP se ocupa al igual que en las otras, un origen, un destino, el grafo, un padre null como en los demás y como elemento auxiliar, una lista de visitados para no entrar en una recursión infinita.
- A diferencia de los anteriores este no necesita una lista de elementos por visitar ya que ya misma forma recorrer los nodos nos ayuda a no dejar alguno libre, por ende, no es necesario la lista de elementos por visitar.
- Al igual que las tres anteriores el método maestro aquí, es expande, el cual se encarga de implementar la búsqueda mediante este método.
- Este expande es bastante sencillo ya que no analiza nada para decidir, solo hay que controlar que no busque en nodos ya analizados.
- Este si es un algoritmo el cual puede necesitar un gran número de recursiones para obtener el resultado lo cual hace que le cueste más encontrar la ruta y al final devuelve rutas que si bien cumplen con lo establecido pueden llegar a ser bastante absurdas.

Búsqueda Primero en Anchura (BPA)

Este algoritmo al igual que el anterior no ocupa información para encontrar el camino al objetivo, es muy similar, pero este en vez de ir profundizando, le da prioridad a los niveles, es decir, primero analiza rodos los nodos del nivel 1, luego los del nivel 2, y así hasta que encuentra el objetivo, de igual forma al no ocupar información puede llegar a ser engorroso, sin embargo la forma en que recorre lo vuelve bastante bueno a pesar de no ocupar información. A continuación que se ocupa para implementarlo en el Código:

- Para ocupar correctamente esta clase variamos un poco a las anteriores, primero igual que en las otra, necesitaremos una instancia del objeto, con origen, destino, papa, grafo y visitados.
- Nuevamente por no ocupar información no necesita un nodo por visitar.

- Este, aunque también cupa una función expande para funcionar esta no debe ser llamada donde sea.
- Para calcular la ruta ocupamos una función llamada recorre niveles la cual recibirá el elemento raíz dentro de una lista de un elemento, ya que al ser la raíz en su nivel solo está el, también se ocupa la lista de visitados para saber a cuáles ir o no ir.
- Ya dentro de la función, recorremos el nivel que nos pasaron como parámetro anteriormente.
- El nivel actual al ser recorrido ira expandiendo y formando a su vez el siguiente nivel, así si no se encuentra el elemento en ese nivel la función se llama recursivamente pero ahora con el siguiente nivel y así hasta que encuentra al elemento y este al regresar trae la ruta con la que se le encontró.

REPARTIR CAMINOS

Una vez explicado como es que los algoritmos nos entregan un camino se procede a explicar como es que se ocupa ese resultado.

Cada uno de los algoritmos esta diseñado para regresar lo mismo sin importar como es que obtengas el camino, aquí es donde el parámetro de padre que se encuentra en cada clase cobra valor, ya que este nos permitirá recorrer el camino desde el objetivo hasta nosotros, eh ahí porque del null como parámetro, ya que así seremos capaces de detenernos cundo llegamos al origen.

Para simplificar todo al recorrer se va generando una lista con los elementos a recorrer para que así sea más fácil la manipulación de la ruta para cada agente.

Ya con las listas de las rutas establecidas, cada agente toma su ruta y empieza a seguir al objetivo.

PERSECUSION

Una vez que cada agente tiene la lista por la cual va a moverse este ocupara la lista como una pila, sacando uno por uno los destinos a los que quiere ir, por ejemplo, si el agente esta en la esquina 4 y va a la 10 pasando por la 13 y 20, su camino quedaría así [10, 20, 13, 4], así empezaría haciendo pop por el final e iría recorriendo de punto a punto.

Para llegar de un punto a otro no es tan simple como solo cambiar la opción, es determinar hacia donde esta la siguiente esquina y entonces ir hacia allí a cierta velocidad, ya cuando llegas a un punto hacemos pop al siguiente y así hasta llegar al objetivo.

CAMBIO DE RUTA

Para este experimento, el objetivo (Kemonito) estará cambiando de posición constantemente por tanto los agentes deben de cambiar su ruta apanas se detecte el cambio, ¿Cómo se logra esto?

Para lograr que los agentes cambien d dirección es algo muy fácil, simplemente se monitorea la posición del objetivo y mientras se mantenga en el mimo lugar los agentes seguirá el camino preestablecido, en cambio si el objetivo cambia de posición, los agentes detendrán su trayectoria hasta que se calcule su nueva ruta, esto en pantalla no se nota mucho, ya que debido al procesamiento parece que simplemente cambian de dirección.

FIN DE LA DEMOSTRACION

Para terminar con la demostración es algo muy simple, los agentes estarán recalculando sus rutas cada que el objetivo lo hace, sin embargo, si algún agente alcanza a tener la misma posición que el objetivo, todos los agentes se detendrán y entonces terminara la demostración.

Así se puede correr varias veces el programa y como la posición del objetivo siempre ser aleatoria, se puede apreciar como cada algoritmo toma una diferente ruta.

Para que la demostración sea más visual, las rutas que ocupara cada agente se pintan de diferente color en el mapa:

- Negro – BPP
- Azul – BPA
- Rojo – A-Estrella
- Verde – Greedy

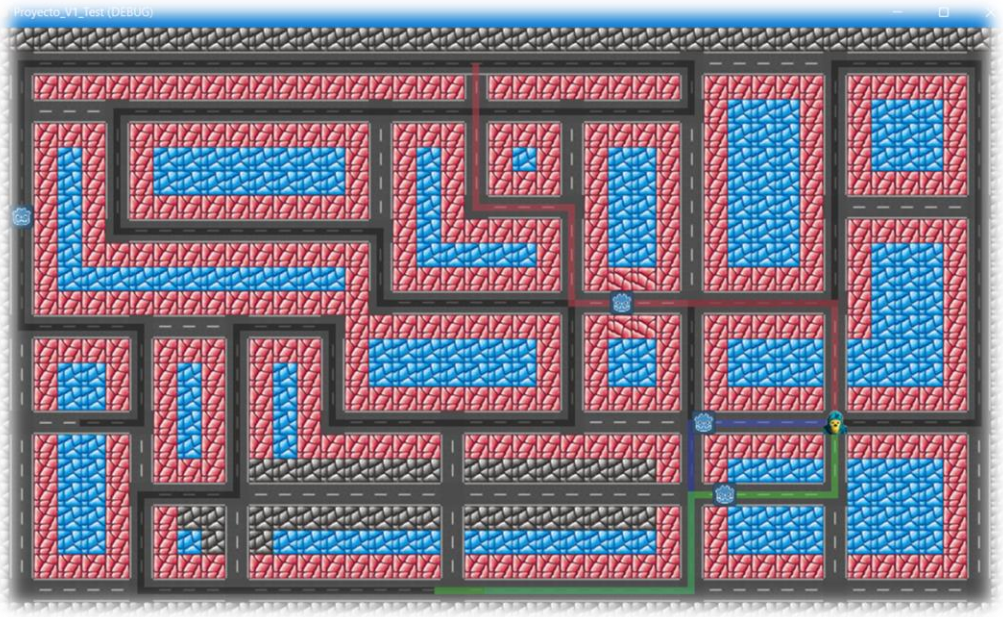

Ilustración 3 : Funcionamiento final de la demostración.

RESULTADOS y CONCLUSIONES

Los resultados fueron bastante claros, creo que se sonó una gran diferencia entre los algoritmos que ocupan información y los que no, aun así, fue bastante sorprendente el resultado del BPA, ya que a pesar de no ocupar información se mantiene muchas veces a la par de los que ocupan información.

Así también fue una decepción total el BPP, los resultados fueron terribles, escoge rutas demasiado malas que tienden a lo absurdo, que, si bien cumple el objetivo, lo hace de muy mala manera, muy inferior a su algoritmo hermano el BPA.

Los resultados que se observaron con los algoritmos de Greedy y A-Estrella fueron los esperados, es bastante difícil notar la diferencia entre uno y otro debido a la forma en que ambos actúan, sin embargo, si se pone atención se puede notar que el A-Estrella es quine mas perfecto es, tomando mejores decisiones en un porcentaje más alto.

Después de todo se puede concluir que cada uno de estos algoritmos funciona de manera correcta, pero también que siempre un algoritmo con información va a entregar mejores resultados, eh ahí el porque de establecer valores y darles peso a las cosas. Nada nos niega ocupar un algoritmo sin información, pero evidentemente es mejor establecer algo que ayude a los agentes a tomar una decisión, y vale mas analizar los problemas para encontrar una heurística que lo solucione que solo implementar un algoritmo simple.