UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA
ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

PROYECTO SEMESTRAL

IMPLEMENTACIÓN DEL ALGORITMO A* EN EL JUEGO LA SERPIENTE

DOCENTE: Chavez Espinoza, William Alberto

ALUMNOS:

- Huancara Ccolqque, Alex Helder 174911
- Quispe Yahuira, Ronaldo 171866
- Sarco Jacinto, Daniel Eduardo 174452

CUSCO - PERÚ

2021

I. TEMA: IMPLEMENTACIÓN DEL ALGORITMO A* EN LE JUEGO DE LA SERPIENTE

II. INTRODUCCIÓN

En una búsqueda para ir de un lugar a otro, generalmente queremos encontrar el camino mínimo entre esos dos puntos(nodos); así como también debemos tener en cuenta el costo en tiempos de viaje. Para encontrar esta ruta mínima de un punto a otro, podemos usar algoritmos de Búsqueda de gráficos, estos algoritmos se usan cuando el mapa se representa como un gráfico. **Búsqueda en anchura**(Breadth First Search) sería la opción más simple dentro de los algoritmos de búsqueda de gráficos. Otra opción sería el algoritmo **Dijkstra**, Y otra sería el **A*** Este es el algoritmo más popular en la búsqueda de gráficos el cual es una mejora del algoritmo Dijkstra. En el presente proyecto trabajaremos con el algoritmo A*.

III. IDENTIFICACIÓN DEL PROBLEMA

- Hallar un camino mínimo de menor coste para que la serpiente controlada por la máquina pueda llegar a su objetivo antes que el jugador humano.

IV. OBJETIVOS

1) Objetivos Generales

- Implementar el algoritmo A* para el juego de la serpiente

2) Objetivos Específicos

- Entender y aplicar el algoritmo A* para resolver el problema planteado.
- Implementar una interfaz para el funcionamiento del algoritmo A* dentro del juego.

V. MARCO TEÓRICO

- **DESCRIPCIÓN DE JUEGO A IMPLEMENTAR**

  El juego de la serpiente (a veces también llamado la serpiente) es un videojuego lanzado a mediados de la década del 70 que ha mantenido su popularidad desde entonces, convirtiéndose en un clásico. En el juego, el jugador o usuario controla una larga y delgada criatura, semejante a una serpiente, que vaga alrededor de un plano delimitado, recogiendo alimentos (o algún otro elemento), tratando de evitar golpear a su propia cola o las

"paredes" que rodean el área de juego. Cada vez que la serpiente se come un pedazo de comida, la cola crece más, provocando que aumente la dificultad del juego. El usuario controla la dirección de la cabeza de la serpiente (arriba, abajo, izquierda o derecha) y el cuerpo de la serpiente la sigue. Además, el jugador no puede detener el movimiento de la serpiente, mientras que el juego está en marcha.

- **LIBRERÍAS A USAR**

Pygame

Es un conjunto de módulos del lenguaje Python que permiten la creación de videojuegos en dos dimensiones de una manera sencilla. Está orientado al manejo de sprites.

Gracias al lenguaje, se puede prototipar y desarrollar rápidamente. Esto se puede comprobar en las competiciones que se disputan en línea, donde es cada vez más usado. Los resultados pueden llegar a ser profesionales.

Características

Pygame permite a los desarrolladores de Python:

- La generación de ventanas gráficas o de pantalla completa con varios tamaños, modos y densidades.
- Tratar la aplicación Pygame como una ampliación de ventanas, donde en un ciclo central se manipula los eventos generados y se actualiza la interfaz gráfica.
- Manipulación y generación de eventos de dispositivos y recursos de la máquina: tarjeta gráfica, teclado, mouse, temporizador, etc.
- La importación, tratamiento y exportación de imágenes desde y hacia ficheros de imagen de formatos conocidos como JPEG, PNG, GIF, TGA, BMP, entre otros, contando el con un tipo propio englobado en la clase Surface. Esto implica algunos efectos de postprocesado (brillo, re coloración, transparencia, máscaras, etc), la deformación (rotación, ampliación y estrechamiento, volteo, entre otras), la obtención de partes de la superficie.

- Generación de formas, líneas y puntos básicos.
- La emisión de sonidos de efecto en ficheros OGG y WAV.
- La reproducción de música de fondo con archivos OGG, MP3 y MIDI.
- Reproducir videos MPEG1.
- Sprites y control de colisiones.

Sys

El módulo provee acceso a funciones y objetos mantenidos por el intérprete.

Random

Este módulo implementa generadores de números pseudoaleatorios para varias distribuciones

Tyme

proporciona un conjunto de funciones para trabajar con fechas y/o horas..

- **HEURÍSTICA**

  En una búsqueda de para determinar el camino más corto entre dos puntos es común encontrar que la frontera se expanda en todas las direcciones buscando el mejor camino posible. Sin embargo, es común encontrar un camino en una sola dirección. Hagamos que la frontera se expanda más hacia la meta que en otras direcciones. Para esa situación usaremos la función Heurística, para indicarnos que tan cerca nos encontramos de la meta.

- **ALGORITMO DE BÚSQUEDA A***

  El problema de algunos algoritmos de búsqueda en grafos informados, como puede ser el algoritmo voraz, es que se guían en exclusiva por la función heurística, la cual puede no indicar el camino de coste más bajo, o por el coste real de desplazarse de un nodo a otro (como los algoritmos de escalada), pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por ello bastante intuitivo el hecho de que un buen algoritmo de búsqueda informada debería tener en cuenta ambos factores, el valor heurístico de los nodos y el coste real del recorrido.

  Así, el algoritmo A* utiliza una función de evaluación $f(n) = g(n) + h'(n)$, donde $h'(n)$ representa el valor heurístico del nodo a evaluar desde el actual,

n, hasta el final, y $g(n)$, el coste real del camino recorrido para llegar a dicho nodo, n, desde el nodo inicial. A* mantiene dos estructuras de datos auxiliares, que podemos denominar abiertos, implementado como una cola de prioridad (ordenada por el valor $f(n)$ de cada nodo), y cerrados, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la $f(n)$ de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados.

El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que $h'(n)$ tiende a primero en profundidad, $g(n)$ tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores.

El algoritmo de Dijkstra funciona bien para encontrar el camino más corto, pero pierde tiempo explorando en direcciones que no son prometedoras. Greedy Best First Search explora en direcciones prometedoras, pero es posible que no encuentre el camino más corto. El algoritmo A* utiliza *tanto* la distancia real desde el inicio como la distancia estimada hasta la meta.

A* usa la heurística para reordenar los nodos de modo que se más probable que el nodo objetivo se encuentre antes

$$F = G + H'$$

F : Costo total del nodo

G : Distancia entre el nodo actual y el nodo de inicio

H' : Distancia heurística estimada desde el nodo actual hasta el nodo final.

- **PSEUDOCÓDIGO A***

ALGORITMO-A-ESTRELLA

```
ABIERTOS := [INICIAL] //inicialización
CERRADOS := []
f'(INICIAL) := h'(INICIAL)
repetir
```

```
        si ABIERTOS = [] entonces FALLO
        si no // quedan nodos
          extraer MEJORNODO de ABIERTOS con f' mínima
          // cola de prioridad
          mover MEJORNODO de ABIERTOS a CERRADOS
          si MEJORNODO contiene estado_objetivo entonces
            SOLUCION_ENCONTRADA := TRUE
          si no
            generar SUCESORES de MEJORNODO
            para cada SUCESOR hacer TRATAR_SUCESOR
hasta SOLUCION_ENCONTRADA o FALLO
```

TRATAR-SUCESOR

```
SUCESOR.ANTERIOR := VIEJO
// coste del camino hasta SUCESOR

caso SUCESOR = VIEJO perteneciente a CERRADOS
  si g(SUCESOR) < g(VIEJO) entonces // (no si monotonía)
  // nos quedamos con el camino de menor coste
  VIEJO.ANTERIOR := MEJORNODO
  actualizar g(VIEJO) y f'(VIEJO)
  propagar g a sucesores de VIEJO
  eliminar SUCESOR
  añadir VIEJO a SUCESORES_MEJORNODO
caso SUCESOR = VIEJO perteneciente a ABIERTOS
  si g(SUCESOR) < g(VIEJO) entonces
  // nos quedamos con el camino de menor coste
  VIEJO.ANTERIOR := MEJORNODO
  actualizar g(VIEJO) y f'(VIEJO)
  eliminar SUCESOR
  añadir VIEJO a SUCESORES_MEJORNODO
caso SUCESOR no estaba en ABIERTOS ni CERRADOS
  añadir SUCESOR a ABIERTOS
  añadir SUCESOR a SUCESORES_MEJORNODO
f'(SUCESOR) := g(SUCESOR) + h'(SUCESOR)
```

OBSERVACIONES:

Como se mencionó anteriormente h’(x) es un estimador de h(x) que informa la distancia do nodo objetivo, entonces:

Si h'(x) hace una estimación perfecta de h(x), A* converge inmediatamente al objetivo.

Si h'(x) = 0, la función g(x) controla la búsqueda.

Si h'(x) = 0 y g(x) = 0 entonces la búsqueda será aleatoria.

Si h'(x) = 0 y g(x) = 1 o constante la búsqueda será Primero en Anchura.

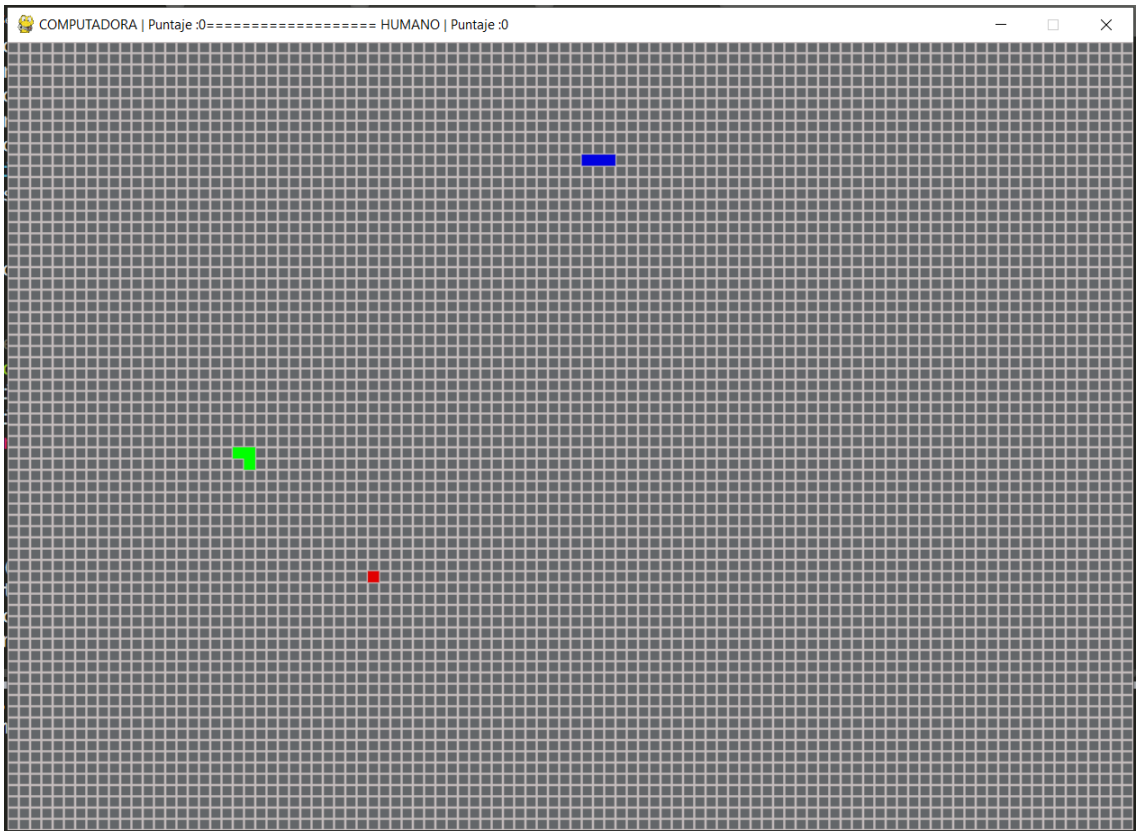
Si h'(x) nunca sobreestima a h(x), (o subestima), se garantiza encontrar el camino óptimo, pero se desperdicia esfuerzo explorando otras rutas que parecieron buenas.

Si h'(x) sobrestima a h(x), no puede garantizarse la consecución del camino del menor coste.

VI. MODIFICACIÓN DEL ALGORITMO A* PARA RESOLVER EL PROBLEMA PLANTEADO

```
Algoritmo_A_Asterisco(Mapa, inicio, objetivo)
  Nodo_incio = inicio
  Nodo_fin = objetivo
  lista_abierta = []
  lista_cerrada = []
  lista_abierta.agregar(Nodo_incio)
  ArregloCostos=0
  while lista_abierta.length > 0
    Nodo_actual = lista_abierta[0]
    for i to lista_abierta.length
      if lista_abierta.f < Nodo_actual.f
        Nodo_actual= lista_abierta[i]
    lista_abierta.remover(Nodo_actual)
    lista_cerrada.agregar(Nodo_actual)
    #Generar El camino
    if Nodo_actual == Nodo_fin
      camino = []
      actual = Nodo_actual
      while actual is not None
        camino.Agregar(actual.posicion)
        actual = actual.pariente
      return camino[::-1] #Inverte la lista de caminos
    #Agrega los todos los sucesores a la lista de sucesores
    sucesores = []
    adyacentes = [(-1, 0), (1, 0), (-1, -1),(1, 1)]
    for pos in adyacentes:
      Nodo_posicion = (Nodo_actual.posicion[0] + pos[0], Nodo_actual.posicion[1] + pos[1])
      if Nodo_posicion[0] > (mapa.length - 1) or Nodo_posicion[0] < 0 or Nodo_posicion[1] > ((mapa[len(mapa).length-1]) -1) or Nodo_posicion[1] < 0
        continuar
      if (not(mapa[Nodo_posicion[0]][Nodo_posicion[1]] in ArregloCostos))
        continuar
      nuevo_Nodo = Nodo(Nodo_actual, Nodo_posicion)
      sucesores.append(nuevo_Nodo)
    # Agrega los sucesores a lista abierta porque aun no han sido visitados
    for sucesor in sucesores
      xAux=sucesor.posicion[0]
      yAux=sucesor.posicion[1]
      sucesor.SetG(Nodo_actual.g +mapa[xAux][yAux])
      sucesor.SetH(abs(sucesor.posicion[0] - Nodo_fin.posicion[0])**2 + abs(sucesor.posicion[1] - Nodo_fin.posicion[1])**2)
      sucesor.SetF(sucesor.GetG() + sucesor.GetH())
      lista_abierta.append(sucesor)
```

VII. JUEGO DE LA SERPIENTE



VIII. CONCLUSIONES

1. El algoritmo A* es unos de los mejores algoritmos de búsquedas en grafos o árboles binarios, de la forma de cómo maneja sus costes la cual este algoritmo es eficaz para el desarrollo de múltiples de juegos de estrategia o defensa.
2. El juego tiene la capacidad de que la computadora pueda jugar en contra nuestra (quitándonos el alimento).
3. Al finalizar la implementación del juego podemos visualizar en la ejecución que la IA usada por la computadora es mucho más eficaz que la capacidad humana, es casi imposible ganar a la computadora ya que calcula la menor distancia para llegar a la comida de forma inmediata.

IX. REFERENCIAS

1. https://n9.cl/9w7dq
2. https://www.python.org/search/?q=typing&submit=
3. https://n9.cl/7rgow

4. https://www.mclibre.org/consultar/python/lecciones/python-biblioteca-random.html