UPC
Universidad Peruana
de Ciencias Aplicadas

Ciencias de la Computación 2022-2

INTELIGENCIA ARTIFICIAL

TB1

Integrantes :

Caballero Lara , Eduardo - U202019644

Wu Pan, Tito Peng - U201921200

Santisteban Cerna, José Mauricio - U201922760

Profesor :

Reyes Silva, Patricia Daniela

2022

Índice

1. Planteamiento del juego

Nuestro proyecto se llama Snake AI, es un juego que recrea el homónimo de 1970 con la particularidad que se juega por sí solo. La serpiente protagonista se mueve por el escenario para comer todas las partículas comestibles hasta crecer y ocupar toda la pantalla evitando chocarse con los obstáculos.

Tomamos como incentivo el campo de los *speedrunners*, donde un jugador profesional se dedica a finalizar un porcentaje de un juego en el menor tiempo posible usando distintas técnicas para lograrlo. Con la implementación de Inteligencia Artificial podemos observar el recorrido autónomo y veloz para lograr el objetivo del juego de forma casi instantánea.

2. Fundamentación del algoritmo

Consideramos más factible utilizar el algoritmo de A* para la implementación del juego. En una versión con jugador, este buscaría el mejor camino para llegar al alimento y evitar los obstáculos. De acuerdo a esto, si queremos que el juego se conduzca por sí mismo, la serpiente protagonista deberá buscar el camino más corto hacia cada alimento y llegar a él. Primero inicializamos los atributos del punto de llegada así como los posibles conjuntos de caminos y direcciones por los que puede moverse la serpiente y donde puede estar el alimento. Asimismo, mediante el algoritmo, se calcula las distancias entre el alimento y el punto de la pantalla donde se ubica la serpiente, se agregan los conjuntos más cortos y luego el personaje se dirige a ellos.

3. Explicación de la heurística

La heurística utilizada para con el algoritmo fue utilizando la distancia Manhattan entre el alimento y el punto inicial de la serpiente. De este modo, se calcula la distancia en línea recta entre dos nodos (posiciones) y se escoge el camino con menor coste, el cual se divide entre el movimiento horizontal y el movimiento vertical.

Para el programa se utilizó la siguiente fórmula para calcular el coste:

$$f = g + h$$

4. Detalles del código fuente de la aplicación

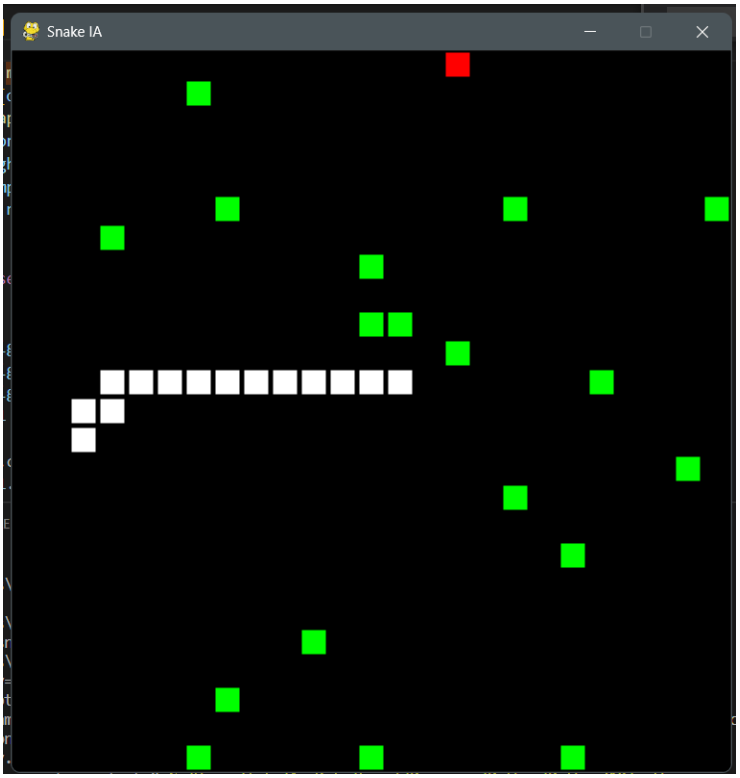
Para la realización del programa se usó la librería Pygame, que se utiliza para creación de videojuegos en 2D de una manera más sencilla y fácil de programar, el módulo randint para generar números aleatorios y numpy para cálculos matemáticos de la distancia. Mediante el

uso de la extensión Mintlify de Visual Studio Code, que usa inteligencia artificial para crear comentarios, se documenta las partes más importantes del código en inglés.

5. Pruebas de uso y descripción de las funcionalidades

A. Menú principal del programa

B. **Programa en ejecución**



C. Inicialización de posición inicial, alimento y obstáculos

```
# Creating a 2D array of Spot objects.
grid = [[Spot(i, j) for j in range(cols)] for i in range(rows)]


for i in range(rows):
    for j in range(cols):
        grid[i][j].add_neighbors()

# Initializing the snake and the food.
snake = [grid[round(rows/2)][round(cols/2)]]
food = grid[randint(0, rows-1)][randint(0, cols-1)]
current = snake[-1]
dir_array = getpath(food, snake)
food_array = [food]
```

D. Algoritmo A* en la función *getpath*

```
class Spot:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.f = 0
        self.g = 0
        self.h = 0
        self.neighbors = []
        self.camefrom = []
        self.obstacule = False
        if randint(1, 101) < 3: self.obstacule = True

    def show(self, color):
        draw.rect(screen, color, [self.x*hr+2, self.y*wr+2, hr-4, wr-4])

    def add_neighbors(self):
        if self.x > 0:
            self.neighbors.append(grid[self.x - 1][self.y])
        if self.y > 0:
            self.neighbors.append(grid[self.x][self.y - 1])
        if self.x < rows - 1:
            self.neighbors.append(grid[self.x + 1][self.y])
        if self.y < cols - 1:
            self.neighbors.append(grid[self.x][self.y + 1])
```

E. Inicialización de valores f, g, h de cada punto del mapa

```
# The A* algorithm.
while 1:
    # Getting the Spot object with the lowest f value from the openset list and adding it to the
    # closedset list.
    current1 = min(openset, key=lambda x: x.f)
    openset = [openset[i] for i in range(len(openset)) if not openset[i] == current1]
    closedset.append(current1)
    # Iterating through the neighbors of the current spot.
    for neighbor in current1.neighbors:
        if neighbor not in closedset and not neighbor.obstacule and neighbor not in snake1:
            tempg = neighbor.g + 1
            if neighbor in openset:
                if tempg < neighbor.g:
                    neighbor.g = tempg
            else:
                neighbor.g = tempg
                openset.append(neighbor)
            # Calculating the distance between the current spot and the food.
            neighbor.h = sqrt((neighbor.x - food1.x) ** 2 + (neighbor.y - food1.y) ** 2)
            neighbor.f = neighbor.g + neighbor.h
            neighbor.camefrom = current1
    if current1 == food1:
        break
# Getting the path from the snake's head to the food.
while current1.camefrom:
    if current1.x == current1.camefrom.x and current1.y < current1.camefrom.y:
        dir_array1.append(2)
    elif current1.x == current1.camefrom.x and current1.y > current1.camefrom.y:
        dir_array1.append(0)
    elif current1.x < current1.camefrom.x and current1.y == current1.camefrom.y:
        dir_array1.append(3)
    elif current1.x > current1.camefrom.x and current1.y == current1.camefrom.y:
        dir_array1.append(1)
    current1 = current1.camefrom
```

6. Referencias bibliográficas

Pygame Documentation. (17 de septiembre de 2022). Pygame.https://www.pygame.org/docs/

Pygame Tutorial. (17 de setiembre de 2022). Introduction to Pygame.
https://pygame.readthedocs.io/en/latest/1_intro/intro.html

Repositorio al código**:**
https://github.com/TitoWuPan/INTELIGENCIA-ARTIFICIAL-TB1.git