Tarea Académica 1 Pacman con Hill Climbing y Simulated Annealing

Nombre de la aplicación: Pacman Escape Simulated Annealing

Presentado por:
Estudiante 1: Daniel Núñez Robinson
Estudiante 2: Diego Bedriñana Alberca

Conceptos básicos necesarios para crear el proyecto:
Lenguaje: Python
Interfaz Web: JupyterLab
Librerías: Graphics.py
Tipo de IA: Hill Climbing adaptado al juego y Simulated Annealing, Temperatura = 20 Rondas
Motivo: Hill Climbing haría que acabe el juego habiendo encontrado el primer máximo local
Heurística: La mínima distancia manhattan posible con respecto a un fantasma y Pacman
Mapa: Una grilla basada en matrices. Paredes (0), Lugares recorribles (1), índices de posiciones X e Y para nuestros 3 actores (Pacman, Fantasma 1 y Fantasma 2)

I. Planteamiento del juego o planteamiento del problema a resolver 300 palabras máximo.

El juego que hemos creado es un Pacman controlado por una Inteligencia Artificial (IA) que decide el mejor camino a tomar para esquivar a los fantasmas en base al algoritmo de Hill Climbing y Simulated Annealing. El videojuego está presentado por una grilla de 11x11 en la cual se van a posicionar 2 fantasmas y a Pacman en distintas posiciones del mapa. La solución es interactiva, ya que los fantasmas son controlados manualmente por aquel que juegue con las el sistema de movimiento por teclas 'WASD', y el reto consiste en que Pacman sobreviva al menos 20 ciclos de movimiento (cuando los fantasmas y Pacman se terminen de mover, esto contará como un ciclo) al utilizar el movimiento de los fantasmas de varias formas.

El problema que buscamos resolver es el de encontrar el camino menos peligroso o más efectivo para Pacman, y esto se puede aplicar en la vida real cuando se generan rutas posibles hacia un destino. Si una ruta contiene estadísticas de accidentes altas o si se puede notar una aglomeración de carros elevada, generalmente es una mejor decisión buscar otra forma de llegar al objetivo.

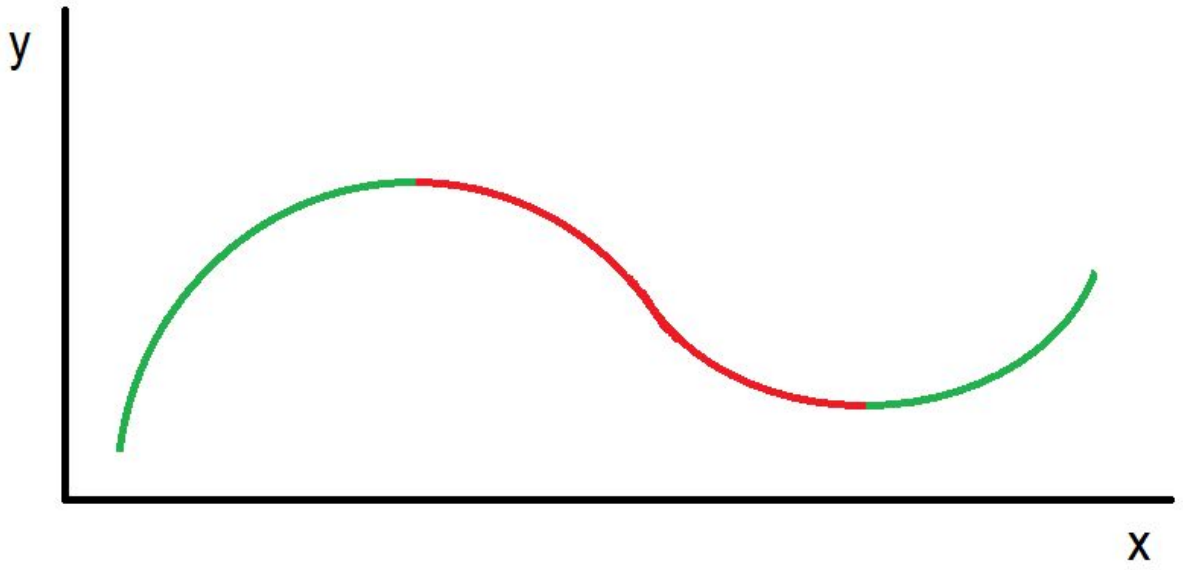
II. Fundamentación de cómo adapta o usa la técnica o el algoritmo de Inteligencia Artificial, explique las entradas el proceso y salida de acuerdo al problema puede usar una gráfica o arquitectura. (300 palabras máximo).

Por la naturaleza similar de estas dos IAs, hemos implementado ambas soluciones Hill Climbing y Simulated Annealing para demostrar qué tan efectivas son cuando se aplican a la situación de escapar del peligro.

El comportamiento del Hill Climbing consiste en que Pacman se quedará quieto una vez que la siguiente heurística conseguida sea menor a la anterior, es decir que se

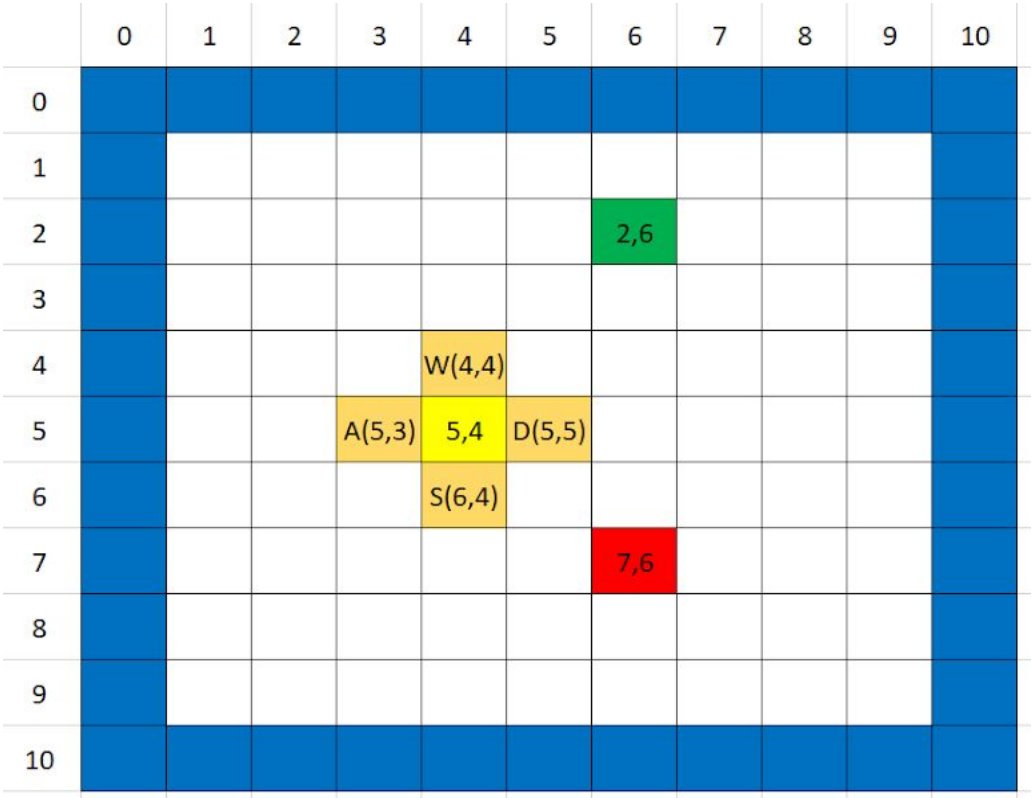
quedará estático una vez que encuentre su máximo local. Aún así decidimos, por motivos de mantener la jugabilidad de nuestro software, que si Pacman encuentra que la heurística sube de nuevo, él comenzará a escapar nuevamente.

Mientras tanto, el comportamiento del Simulated Annealing consiste en forzar el movimiento de Pacman a pesar que ya se llegó a un máximo local, así esperando encontrar un siguiente máximo durante el recorrido de la temperatura. Cabe mencionar que el cooling rate es de 20 rondas, es decir de 20 ciclos de movimiento (cuando los 3 personajes se mueven).



La heurística aplicada (explicada más adelante) nos permite hallar que fantasma está más cerca de Pacman en cierta posición del mapa.

Primero, comparamos todas los posibles lugares a los cuales Pacman puede moverse (arriba, abajo, izquierda, derecha) y le asignamos a esa posición el valor de la distancia del fantasma más cercano (distancia menor).



Teniendo estos datos, escogemos movernos a la posición que nos aleje más de los fantasmas; es decir, la posición que tenga el valor mayor. En caso que las heurísticas calculadas en el ciclo no sean mejores que la anterior, Pacman se quedará en su posición actual y a esto se le considerará un máximo local.

	distancia manhattan			
	dist = |(x1 - x2)| + |(y1 - y2)|			
Posición	Distancia a Verde	Distancia a Rojo	Mas cercano	Nos movemos a
W	4	5	Verde **(4)**	**A (5)**
A	6	5	Rojo **(5)**	
S	6	3	Rojo **(3)**	
D	4	3	Rojo **(3)**	
	Nos quedamos con el menor		*Nos quedamos con el mayor*	

Limitaciones de Hill Climbing: Efecto 'hacerse bolita'

Un problema que encontramos al utilizar Hill Climbing se presenta cuando los fantasmas se aproximan a pacman por dos lados distintos, ya que desde un inicio la mejor heurística fue

la inicial. Esto causa que Pacman se quede quieto mientras que es acorralado por los fantasmas.

III. Explicación de la heurística y de qué manera permite resolver el problema en cuestión (300 palabras máximo).

La heurística seleccionada es la distancia mínima entre Pacman y los fantasmas. Lo que permite calcular a qué posición es más conveniente moverse entre las posibles opciones.

Para lograr esto, decidimos usar la formula de distancia Manhattan, pues consideramos ser la más adecuada dado que estamos usando un sistema de grilla.

Distancia Manhattan = |(x1 - x2)| + |(y1 - y2)|

Aplicando esta heurística, Pacman es capaz de hallar cuál de sus posibles movimientos lo aleja más de los fantasmas de una manera muy veloz, ya que se está trabajando con los índices de los fantasmas y no se tiene que hacer ningún recorrido de la matriz que pueda ralentizar el juego.

Una limitación que encontramos al momento de usar esta heurística es que no estamos calculando la distancia real necesariamente (con un path finding debido a las paredes) y esto puede generar una heurística imperfecta en cuanto a qué es realmente seguro y qué no cuando pacman analiza sus posibles movimientos.

IV. Detalles del código fuente de la aplicación: si ha utilizado algún framework, especificar librerías si fuera el caso que está usando, API etc, todo lo que fuera necesario para poner en marcha su aplicación.

Para el desarrollo del juego, se utilizó la interfaz web JupyterLab y el lenguaje Python. En cuanto a librerías, se utilizó graphics.py para la interfaz gráfica. Fuera de esto, no fue necesario utilizar algún otro software externo.

V. Pruebas de uso, ejecución y descripción de las funcionalidades (3 a 5 capturas de pantalla)

Funcionalidad 1: Condiciones de Victoria/Pérdida

Para que los fantasmas consigan la victoria, deben acorralar o colisionar con Pacman antes que terminen las 20 rondas. Para verificar esto, se verifica que la heurística final encontrada sea 0 o que los índices de posición de Pacman y uno de los dos fantasmas sea la misma. Por otro lado, solo debe recorrer el índice 20 veces para que Pacman gane; es una simple condicional.

Funcionalidad 2: Control de Rondas

En la parte inferior izquierda de la interfaz, mostramos la ronda actual en la que se encuentra el juego. Esto es para saber qué está sucediendo con el algoritmo usado en dicha ronda. Para poder calcular la ronda, usamos un índice que se va incrementando tras cada ronda.

Funcionalidad 3: Dibujo de grilla y personajes más su movimiento

A partir de los índices de posición de los fantasmas y Pacman, se dibuja en cada ciclo de movimiento la posición actual de cada personaje. Esto se va actualizando constantemente con las funciones de movimiento y dibujo implementadas. Los índices de posición de los fantasmas se determinan en base a los movimientos que elijan los jugadores en base a las llaves 'WASD' y, a partir de esto, se modifican los índices de posiciones.

Funcionalidad 4: Cálculo de Heurística

Se calcula la heurística de distancia manhattan manejando los índices de posición de los fantasmas y Pacman. Entre los dos fantasmas, el número mínimo es el que se guarda para la posición seleccionada. Una vez que se calculan todas las heurísticas, se elige la que tenga un número mayor para que Pacman pueda escapar de la mejor manera.

Prueba de Uso 1:

Detonar la condición de pérdida una vez que un fantasma conecta con Pacman. Comportamiento esperado: Que los condicionales de colisión presenten False True o True False o True True y que el juego lo procese.

Prueba de Uso 2:

Detonar la condición de pérdida una vez que los fantasmas acorralan a Pacman. Comportamiento esperado: Si la mejor heurística calculada es 0, que el juego determine esto como el final del juego. Estado es victoria para fantasmas.

Prueba de Uso 3:

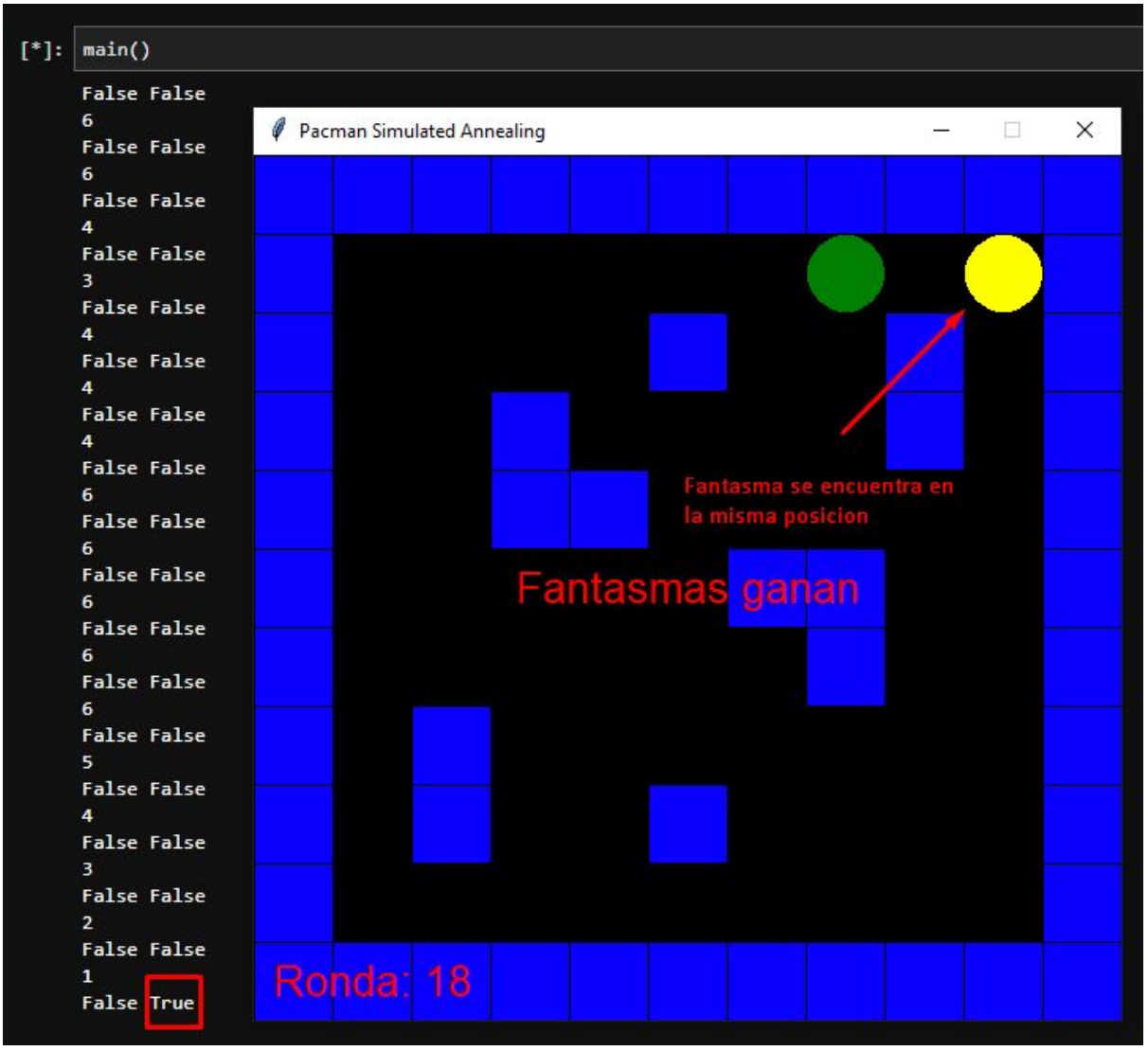
Detonar la condición de victoria una vez que pasan las 20 rondas y no se ha perdido. Comportamiento esperado: Una vez que el conteo de rondas llegue a 20, el juego debe acabar como victoria para Pacman

```python
def heuristica(pacman):
    global pacman_x
    global pacman_y
    global fantasma1_x
    global fantasma1_y
    global fantasma2_x
    global fantasma2_y

    move_to = ''
    global best

    #Queremos saber si pacman chocó con un fantasma para determinar que perdió
    actual1 = pacman_x == fantasma1_x and pacman_y == fantasma1_y
    actual2 = pacman_x == fantasma2_x and pacman_y == fantasma2_y

    #Si es así, se ejecuta la condición de perdida
    print(actual1,actual2)
    if actual1 == True or actual2 == True:
        return False
```

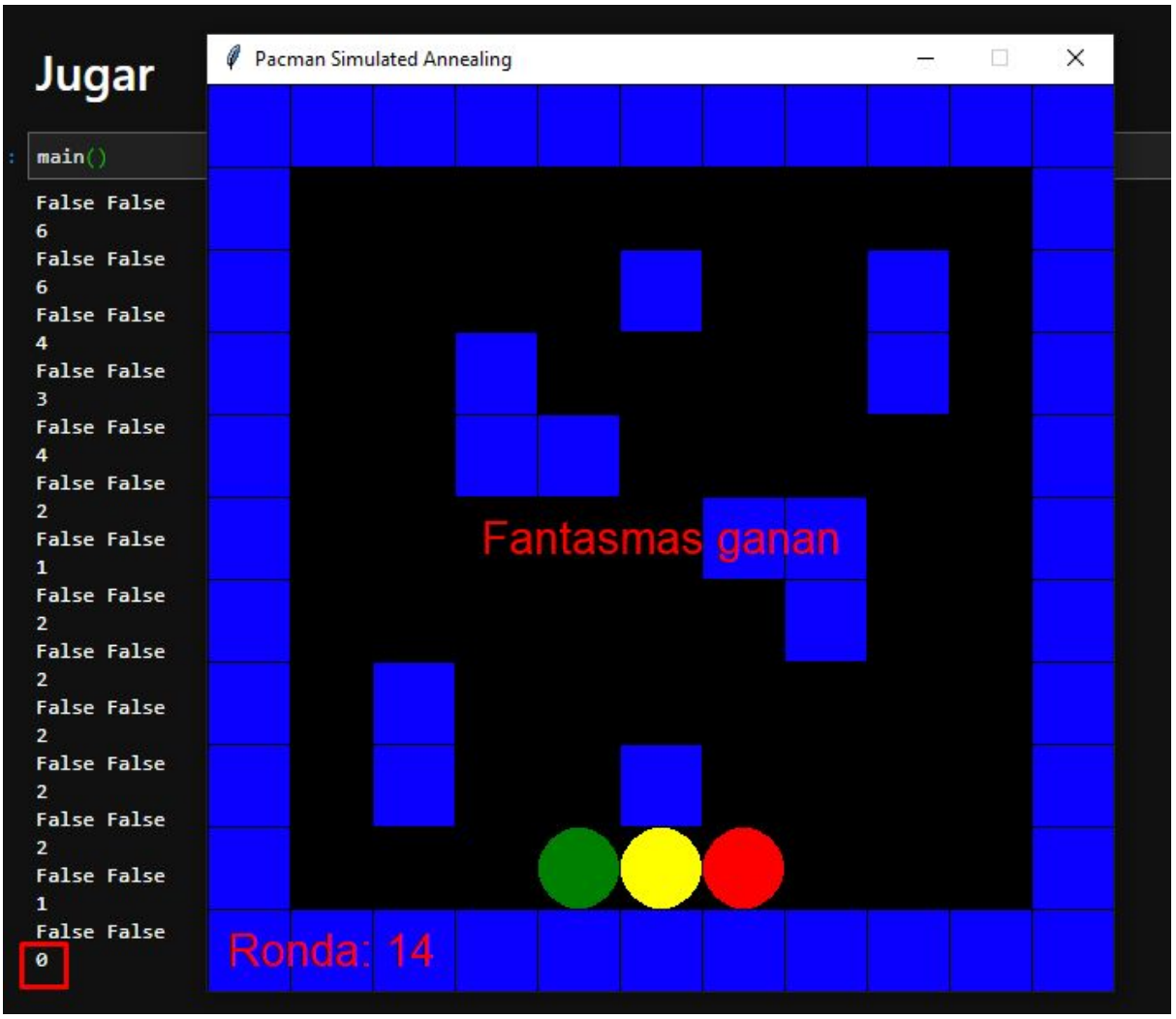
Cálculos para la colisión entre Pacman y un fantasma para ejecutar condición de pérdida

```python
  mejor_izquierda = -1
  if matrix[pacman_y][pacman_x - 1] != 0:
      izquierda1 = abs((pacman_x - 1) - fantasma1_x) + abs(pacman_y - fantasma1_y)
      izquierda2 = abs((pacman_x - 1) - fantasma2_x) + abs(pacman_y - fantasma2_y)
      mejor_izquierda = min(izquierda1, izquierda2)
      if best <= mejor_izquierda:
          best = mejor_izquierda
          move_to = 'a'

  mejor_derecha = -1
  if matrix[pacman_y][pacman_x + 1] != 0:
      derecha1 = abs((pacman_x + 1) - fantasma1_x) + abs(pacman_y - fantasma1_y)
      derecha2 = abs((pacman_x + 1) - fantasma2_x) + abs(pacman_y - fantasma2_y)
      mejor_derecha = min(derecha1, derecha2)
      if best <= mejor_derecha:
          best = mejor_derecha
          move_to = 'd'


  best = max(mejor_arriba,mejor_abajo,mejor_izquierda,mejor_derecha)
  print(best)
  if best < 1: #Pacman pierde
      return False
  else:
      #Juego continúa, pacman sigue escapando
      move_pacman(move_to, pacman)
      return True
```

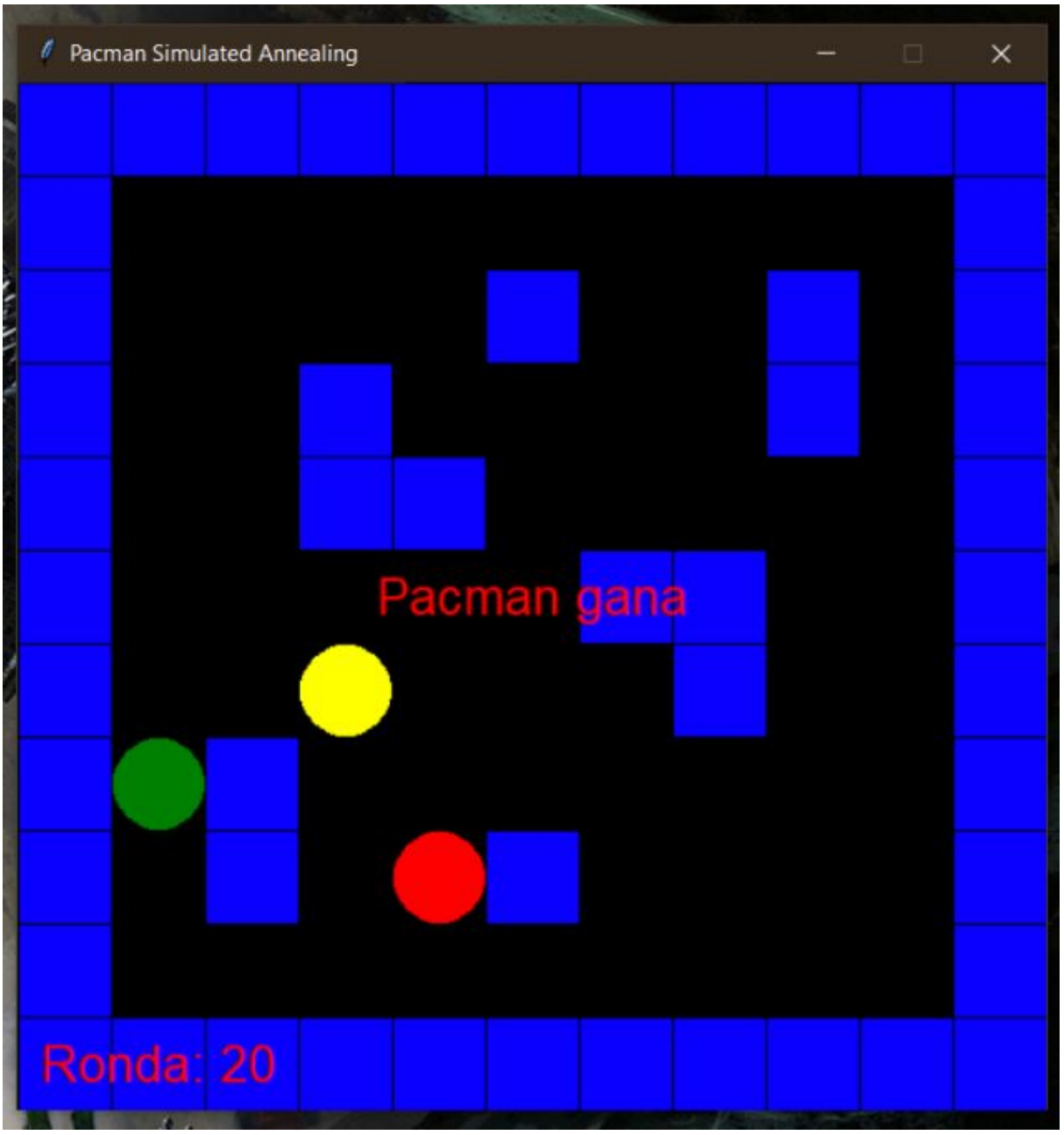
Cálculos de heurística para cada posición más validación para pérdida de Pacman o permitir que se mueva. Se mide la distancia manhattan.



Se evidencia el control de rondas, la medida de heurísticas, el control de colisión entre los fantasmas y Pacman y que se active la condición de victoria para los fantasmas.



Se evidencia que la heurística es 0, es decir que Pacman no tiene otra opción de movimiento restante, por lo que se detona la condición de pérdida.



Se evidencia que el número de rondas transcurridas son 20. Esto significa que la IA ha logrado ganar al escapar por suficientes ciclos de movimiento.

VI. Referencias bibliográficas

John Zelle. (2016). Graphics.py. Recuperado el 4 de abril de 2020 de sitio web: https://mcsp.wartburg.edu/zelle/python/graphics.py

Project Jupyter. (2020). JupyterLab Documentation. 4 de abril de 2020, de Project Jupyter Sitio web: https://github.com/jupyterlab/jupyterlab/blob/ccfc4d9cde8ff75cf24cbba214d937ef98efa6cb/docs/source/index.rst

Python. (2020). Python Documentation. 20 de abril de 2020, de Python Sitio web: https://docs.python.org/3/