PRÁCTICA 2:
DESARROLLO DE JUEGOS CON INTELIGENCIA ARTIFICIAL

Hugo Camuñas Gonzalez, Raúl Delgado Alcázar, Álvaro Codorníu Alonso|GrupoH

1. INTRODUCCIÓN

El objetivo de esta práctica es implementar un agente inteligente en Unity que, mediante el uso del algoritmo de aprendizaje por refuerzo Q-Learning, aprenda a huir de un enemigo. El agente debe entrenarse a partir de la experiencia en distintos escenarios del juego, utilizando una tabla Q para tomar decisiones.

2. DISEÑO DEL ALGORITMO DE Q-LEARNING

2.1 Estados

Cada estado se representa por 8 valores booleanos, lo que en principio daría 256 (2^8) combinaciones posibles:

canMoveNorth: El agente puede moverse al Norte.
canMoveSouth: El agente puede moverse al Sur.
canMoveEast: El agente puede moverse al Este.
canMoveWest: El agente puede moverse al Oeste.
fromNorth: El enemigo está por encima del agente.
fromSouth: El enemigo está por debajo del agente.
fromEast: El enemigo está a la derecha del agente
fromWest: El enemigo está a la izquierda del agente

2.2 Acciones disponibles

Up (0), **Down** (1), **Right** (2), **Left** (3)

2.3 Función de Recompensa

-100 si el agente es atrapado.

-10000 si intenta un movimiento inválido (hacia un muro o fuera del tablero).

+50 si la nueva posición aumenta la distancia Manhattan al enemigo.

-50 si la nueva posición disminuye la distancia Manhattan al enemigo.

Durante el modo de entrenamiento, estas recompensas se usan para actualizar la tabla Q mediante la fórmula de Q-Learning. Todos permanecen fijos, salvo epsilon, que desciende desde 1 a 0,1 a lo largo de los episodios. Para una mejor optimización, se ha optado por actualizar la tabla cada 10000 episodios, siendo la media de episodios totales de cada entrenamiento 100000.

3. ESTRUCTURA DEL CÓDIGO

3.1 MyTrainer

La clase MyTrainer se encarga de entrenar al agente a huir del enemigo usando Q-Learning. Cuando se inicia, recibe los parámetros de aprendizaje (alpha, gamma), la política de exploración (epsilon, que va disminuyendo episodio a episodio), el número total de episodios y cada cuánto debe guardar la tabla Q. Además, toma como referencia el mundo con WorldInfo y el algoritmo que usa el humano para desplazarse. A continuación, carga una tabla de valores anterior si existe, o crea una nueva si no existe dicha tabla, y sitúa al agente y al perseguidor en posiciones aleatorias.

En cada paso de la simulación comprueba si el enemigo ha atrapado al agente, si esto es así, da por terminado el episodio y pasa al siguiente. Si no lo atrapa, ajusta la probabilidad de explorar acciones nuevas o de explotar. Después elige una dirección (norte, sur, este u oeste), calcula la recompensa correspondiente (penalizando o recompensando) y actualiza la tabla Q con ese dato.

3.2 MyTester

Una vez entrenada y guardada la Q-Table, su único propósito es leer ese CSV, reconstruir el diccionario de estados a valores Q. En cada frame de la simulación, decide la acción óptima para el agente sin realizar aprendizaje.

En cada llamada a GetNextStep, toma la posición actual del agente y la del enemigo, codifica el estado como los ocho booleanos, y recupera los cuatro valores Q asociados a ese estado. Luego emplea una función para elegir la acción con el valor más alto y la traduce a una dirección como Norte, Sur, Este u Oeste. Finalmente, devuelve la celda a la que se mueve el agente.

4. TABLA Q

La tabla usa ocho booleanos para definir cada estado. Los cuatro primeros indican hacia dónde puede moverse el agente, true = transitable, false = obstáculo. Los otros cuatro señalan la posición relativa del enemigo.

Por combinatoria, la tabla resultante tiene 256 estados, los cuales se han inicializado con valores arbitrarios para distintas acciones:

- -99999 para las acciones en las que se interponga un muro.
- 0 para el resto de acciones.

Esto es una decisión de diseño que busca agilizar el aprendizaje y evitar de entrada acciones que son demasiado perjudiciales o imposibles para el agente, penalizándolas desde el inicio para que no las explore ni las aprenda como válidas.

En el contexto de esta práctica, el peor caso es intentar atravesar un muro, ya que eso lleva al agente a quedarse quieto hasta que es atrapado, en caso de quedarse bloqueado por 3 direcciones y acercarse el jugador por la restante.