Escuela Técnica Superior de Ingeniería Informática

3º Grado en Diseño y Desarrollo de Videojuegos

Móstoles

Desarrollo de Juegos con Inteligencia Artificial

Práctica 2: Machine Learning

Implementar un agente inteligente cuyo comportamiento sea aprendido por técnicas de machine learning, en concreto Q-Learning en Unity

Pablo García Núñez

Isabel Villoria López

1. Contexto del juego

La práctica que se ha desarrollado comprende un tablero de 20x20 unidades donde un Zombie (el agente) debe huir del Player. Esto se hará mediante un aprendizaje previo del Zombie mediante un algoritmo de Q-Learning que luego utilizará en distintos escenarios para aguantar el máximo tiempo posible lejos del otro personaje.

2. Problema que se plantea a la Inteligencia Artificial

El agente debe huir de un jugador que le busca en un tablero mediante un algoritmo de búsqueda A*. Para hacerlo, deberá aprender en un escenario de prueba las distintas casuísticas que se le pueden presentar para elegir entre ellas la más beneficiosa para su objetivo.

Este proceso de aprendizaje (machine learning) se ha desarrollado mediante un algoritmo de Q-Learning donde el agente posee una tabla Q con valores que le informan sobre las distintas acciones que puede realizar en ellos. Para el proceso de aprendizaje deberá encargarse de rellenar dicha tabla Q con los valores que vaya calculando mediante la siguiente fórmula:

$$Q'(s,a) = (1 - \alpha)\ Q(s,\ a) + \alpha(r + \gamma\ \max Q(s\ ',\ a\ '))$$

Una vez que la tabla estuviese rellenada y pudiese proporcionar conocimiento al agente sobre el mundo y sus posibles acciones según la recompensa que recibía por moverse a distintos estados, se podía pasar a la fase de prueba. En esta, se empezaría el juego para poner a prueba el conocimiento recopilado en la tabla Q.

3. Algoritmos implementados y decisiones tomadas

El primer paso que se ha seguido para solucionar la problemática presentada fue definir el modelo que seguir. Tras comprender las interfaces y la base existente, se procedió a desarrollar una clase MyQStates que contuviese toda la información que concierne el mundo.

Los posibles estados en los que el agente se podía situar podían abarcar infinitos factores, por ello, se decidió acotar a tres parámetros: distancia entre el agente y el otro, orientación entre ambos y el estado de las celdas colindantes a la actual (si son Walkable o no). Además, se discretizaron dichos atributos para que el número de estados fuese controlado y acotado para un mejor funcionamiento dentro de la tabla Q.

La distancia se acotó como si fuese un rango de expansión de 0 a 2, siendo 0 donde la distancia (Manhattan) entre ambos sería un valor menor o igual a 5 y 2 cuando este valor superase 15. De esta forma, en vez de 40 posibles combinaciones se redujo a 3 que funcionaban bien con el juego. La orientación se discretizó en 8 valores (de 0 a 7) que valoraban la distancia relativa entre ambos personajes. Limitando los 360º en 8 valores mediante el uso de la tangente para un mejor uso. Finalmente, los estados de las celdas colindantes se basaron en 4 booleanos que barajaban todas las combinaciones posibles sobre si era una celda con muro/fuera de mapa o no. De esta forma, quedarían un total de 384 estados ($3 * 8 * 2^4$).

Una vez definido el estado, se tuvo que decretar el funcionamiento de la tabla Q, la cual guardaría la información sobre la bondad de cada acción dependiendo del estado. Para guardar los datos, lo primero fue crear un diccionario cuya clave fuera nuestra clase MyQStates (se tuvo que sobrescribir los métodos Equals y GetHashCode) y el valor fuera un array de floats que guardaría los valores Q. De esta forma, se podía guardar y acceder a los datos de la tabla Q de manera eficiente. Cuando se crea la tabla por primera vez, se debe llamar después al método InitializeQTable para inicializarla con un valor de -100 donde la acción no fuese accesible (no Walkable) y 0 en cualquier otro caso. Además, se crearon métodos para guardar la tabla en formato CSV mediante un StringBuilder; para cargar la última tabla creada y rellenar el diccionario con sus valores; y una simple función para determinar todos los posibles valores del cellState.

Finalmente, en cuanto al sistema de recompensas tuvimos que realizar varias iteraciones hasta encontrar las que mejores resultados nos han dado. Empezamos teniendo varias recompensas según los atributos del estado siguiente y, aunque estos valores parecieron funcionar durante bastantes pruebas, de vez en cuando fallaban. Por ello acabamos limitándolas a las siguientes:

- Si la siguiente celda no es Walkable: -100
- Si la siguiente celda es donde está el otro: -100
- Si la distancia desde la nueva celda es mayor que la inicial: +10
- Si la distancia desde la nueva celda es menor que la inicial: -1
- Si la distancia desde la nueva celda es igual a la inicial: 0

A la hora de programar el Trainer, se utilizó la interfaz IQMindTrainer completando sus métodos. En Initialize se inicializaron los parámetros de entrada, se creó la tabla Q o se cargaba una ya creada y se asignaban los

parámetros del algoritmo (dichos datos se comentarán en su respectivo apartado). En el método DoStep se debía programar un paso del entrenamiento. Para ello, lo primero era iniciar el episodio si no se había hecho e iniciar los contadores necesarios. Después, se determinaba el estado actual, discretizando los parámetros correspondientes a la posición del Agente y el otro en dicho momento.

El siguiente paso era elegir una acción de forma balanceada entre una aleatoria y la mejor para el estado actual. Para esto se hizo uso de la épsilon comparándolo con un número aleatorio. Al principio, para rellenar la tabla y que optara más por acciones aleatorias pues la tabla estaría vacía, se forzó que optara por estas con una variable que iba incrementando lentamente y se iba reiniciando tras dejar paso a un par de mejores acciones.

Con la acción elegida, se ejecutaba, creando el estado siguiente basándose en los resultados obtenidos. Con este nuevo estado y la recompensa correspondiente, se calcula el nuevo valor de Q que tendrá el estado actual mediante la fórmula de Q-Learning expuesta antes. Tras los cálculos y su posterior asignación, se evalúa si ha habido una recompensa de -100 (el agente ha ido a una celda no transitable o ha sido pillado) y se reinicia el episodio si ha ocurrido. Además, al llegar a un cierto número de episodios, se guardan los resultados del diccionario en la tabla Q. Finalmente, se pasan tanto el agente como al otro a la celda del siguiente estado.

Todo este proceso se ha ido compaginando con repetidas pruebas para ir perfeccionando los parámetros del algoritmo y modificando el escenario de entrenamiento y de prueba para un mejor aprendizaje del agente.

4. Controles de los algoritmos y sus valores

Para el aprendizaje se han ido variando los parámetros utilizados según se han ido necesitando.

- **Alpha**: Se ha mantenido con un valor de 0.2, aunque hicimos pruebas con distintos valores como 0.3 y 0.4. Con este valor hemos conseguido un buen ritmo de aprendizaje de forma paulatina que nos ha servido para tener en cuenta la experiencia de forma controlada.
- **Épsilon**: Lo hemos ido modificando según el paso de episodios para que fuese a por acciones mejores según se habían aprendido nuevas cosas. Por

ello, varía entre un 0.85 inicial, y se va reduciendo paulatinamente hasta un 0.3 cuando llevase 200.000 episodios.

- **Gamma**: Aunque hemos probado con valores como 0.7 y 0.8, al final lo hemos dejado en 0.9 para darle más peso a la recompensa, pues nuestras secuencias eran largas y no nos importaba que tardase más en aprender.
- **Episodios totales**: lo hemos subido a 300.000, aunque normalmente lo parábamos antes para hacer cambios en el escenario e ir probando los resultados.
- **Episodios para guardado**: lo hemos fijado a 20.000 para que cada tabla pudiese tener información suficiente para compararlas entre sí.

5. Instrucciones del programa

Para la parte de entrenamiento (escena TrainPlayGround), en el GameObject QMindTrainer se debe cambiar el Navigation Agent al desarrollado por nosotros llamado: MyQMindTrainer. Una vez eso esté actualizado, al ejecutar el agente irá entrenando episodio tras episodio, pudiendo mostrar la simulación o no si se activa/desactiva desde el inspector. Lo óptimo es parar su ejecución (si se quiere antes de los 300.000 episodios) cuando se hayan pasado las 20.000 épocas sobre las que se hace un guardado. Cuando esto ocurra, en la consola aparecerá un pequeño mensaje "GUARDANDO" para avisar de que se ha creado un archivo csv. También, en la esquina superior izquierda saldrá el número de pasos y episodios, la recompensa media y la total.

Para la parte de prueba (escena TestPlayGround), de nuevo en el GameObject QMindTester se debe cambiar el QMindAgent a MyQMindTester y asegurarse que el Agent y el Oponent estén correctamente asignados. Una vez todo esté listo en el escenario a probar, se ejecutará y en la esquina superior izquierda se podrá llevar un conteo de los pasos hasta que el agente es pillado y la ejecución es detenida. Además, por consola saldrá el número de pasos dados y el path de la tabla que está utilizando. Asegurarse de que el archivo csv no está abierto cuando se ejecute, pues ocasionará problemas. Finalmente, el objetivo se verá cumplido cuando el Agente consiga huir del Oponente por mínimo 250 pasos (número que se verá afectado por el entrenamiento realizado o por la tabla Q utilizada).

En ambos casos se puede modificar la velocidad de Agente y Oponente para acelerar el proceso.

6. Conclusiones

El resultado del desarrollo de la práctica ha sido una tarea ardua pero gratificante. Se ha conseguido tras muchas pruebas un resultado óptimo de la huida del agente, llegando a alcanzar más de 10.000 pasos en la mayoría de las ejecuciones. Se debe mencionar que en algunos escenarios con multitud de muros el Agente se quedaba atascado y llegaba a ser pillado por el agente con menos pasos de los deseados. Sin embargo, para la posible utilización en un juego real, los conocimientos obtenidos mediante la realización del algoritmo pueden ser de gran ayuda.

Se ha entrenado al agente por más de 420.000 épocas en distintos escenarios (de distinta complejidad), quedándonos con la tabla 21 que es la que mejor resultados nos ha dado con pruebas como la siguiente: